下载APP

11 | 反爬虫概述（三）：前端反爬虫是怎么做信息收集的？

2022-02-21 DS Hunter

《反爬虫兵法演绎20讲》 课程介绍 >

讲述：DS Hunter

时长 15:54 大小 14.58M

你好，我是 DS Hunter。

上一讲，我们已经了解了前端反爬虫中最主要的动作，key 的加密，但是除了这件事，前端还可以完成更多的细节来辅助整个反爬虫动作。

例如信息收集，它们虽然不会直接完成“拦截”的动作，看起来也没有什么贡献，但是对于工程师的生存以及“拦截”动作本身的指导和优化来说，却有着不小的辅助功能。最后，我们会把 key 的加密、信息收集这一切都聚集到规则引擎中统一收口。

这一讲，我们就把目光聚集在信息收集以及规则引擎这两大辅助部分，完成前端反爬虫的主要工作。

信息收集

我在01 讲中，就强调过信息一直是战争的核心，而信息收集是反爬虫的重中之重。公司有爬虫的需求，是希望在信息战中取得胜利，进而赢得商业的战争。而反爬虫方“收集信息”需求的产生，是由于工程师们想在反爬虫这场战争中打赢。

那么，到底要收集什么信息、怎么收集信息呢？后续我们又要怎么利用这些信息呢？我们按照反爬虫的动作顺序，先从信息预收集的部分开始说起。

信息预收集

在信息预收集模块，我们需要收集用户特征用于分析，而收集这些信息的方法，都可以认为是收集模块。当然，因为用户特征信息会被送到服务端进行分析检测，所以这里我们收集了哪些数据，应该是一个秘密，尽可能让爬虫方不知道才好。这和后面我们提到的“埋点统计分析”中的普通埋点是不同的。普通埋点，不怕爬虫方知道。

我们在《05 | 通用且基本的检测规则》里面说到的浏览器特征以及业务特征，其实都是前端的用户特征检测点。这里，我们主要聚焦在技术方面的浏览器特征分析。

浏览器特征，核心其实就是 DOM 指纹。我们都知道，DOM 是一棵树，所以每个节点都可以通过 XPath 来访问到。同时，还有一部分对象放在了 BOM 上面，也就是通过 window 来直接访问。那么具体的访问方式是什么呢？我来帮你回顾一下。这里我们先规定一下，虽然浏览器 BOM 和浏览器 DOM 是两个东西，但是访问方式都差不多，在下面同时出现的时候，我们统称 DOM 了。虽然这样在学术上不正确，但是其实并不影响我们交流。

window 是我们整个 DOM 对象的根节点，我们从这里出发，理论上可以到达 DOM 对象的任何一个节点。理论上说，我们也可以用 Xpath 来访问任意一个 DOM 节点。部分 BOM 对象即使不可达，我们也可以仿照 XPath 给 BOM 做一个类似的定位符，来标记访问信息。这样，我们从 window 出发，就可以遍历所有的对象了。

但是要注意的是，整个树是存在循环引用的，我们为了避免递归调用死循环，必须进行重复节点检测。这个具体做法就不 demo 了，检测重复节点可以说是经典面试题了，属于基操。

当然，我们不但可以通过上述的过程拉取整个 window 对象的信息，还可以进一步进行特征提取，例如取 window 对象的 MD5——当然记得加盐，直接取 MD5 很容易被猜出来。

我们还可以指定部分的 XPath，然后让这部分节点聚合在一起成为一个对象，在浏览器端进行特征提取。这样我们就可以拿到一个用户指纹了。这个指纹可能防冲突并不好，但是伪造难度也并不低。

我们之所以这么费事，非要提取一个指纹出来，而不是直接把信息上报，有以下几个原因：

1. **被破解几率低：**直接上报很容易被抓，而且因为是明文的，对方一旦知道了逻辑，伪造难度就下降了。
2. **保密程度高：**指纹是加盐的，因此相同的信息，其实上报的 key 是不一样的。但是服务端知道原始 key 是什么，虽然 MD5 不可逆，但是可以通过已有的 key 进行再运算。至于盐的上报，可以夹杂在 MD5 结果里。不一定是 insert，也可以是 replace，因为少验证几位没什么大事。
3. **速度快、压力小**：获取指纹之后，传输数据量会大大降低。这样一方面能提升速度，一方面能降低服务器压力。

误伤统计收集

误伤统计收集则与普通信息不同。如果说信息预收集，信息来源是用户产生的，后面的埋点信息是反爬系统产生的，那么误伤统计则是一个中间地带。因此，信息预收集需要强加密，埋点统计需要弱加密，而误伤统计则落于两者之间，可以随意选一个加密强度。**当然了，这里的所有强、中、弱都是相对而言的，每一个部分的信息都至关重要。**

我们在09 讲提到的误伤统计理论中讲过一个“打点法”，也就是客户端每次访问写入一个 Cookie，然后在订单之类的页面进行拦截。这一讲的误伤统计收集，同样使用这个方法。不过这一讲我们会就它的加密强度来讨论。由于拦截的过程中是直接读取 Cookie 的，那么加密主要就是放在写 Cookie 的地方了。

低级别的加密，是在 Cookie 里加入用户信息。再升级就可以增加不可逆加密，以服务端有办法鉴别有效性为标准。注意，我们提到不可逆加密的时候，有的朋友经常认为，不可

逆加密就是不可逆的，所以客户端肯定不能用，但其实这是错误的想法。

如果在服务端验证时，我们的需求是解密出来的，那不可逆加密当然很难使用。如果我们只是鉴定它是否有效，或者已知入参是有限的，那么就可以使用不可逆加密了。

误伤统计收集与埋点不同，这个不会有高并发——如果有高并发岂不是你的误伤已经血流成河了。因此不用担心高并发无法处理的事情。

不过，如果出于谨慎考虑，也可以做一个熔断：当系统承接不了网络流量时，直接断掉误伤统计，同时熔断掉反爬需求即可。此外，还可以做成更谨慎的：反爬虫默认是断掉的，误伤统计系统定期给爬虫系统充值时间，这样一旦误伤统计系统跪了，反爬虫系统会自动回到默认值，也就是关闭状态。

埋点统计分析

如前面所说，埋点信息是非关键信息，因此可以不用特别强的加密。可能你会问：埋点统计是弱加密，那么，无加密不行吗？

答案是，埋点统计可能也会被抓取，也就是爬虫可能爬取埋点，来影响你的统计和决策。不过，由于要得到这个效果的过程是很漫长的，所以大部分爬虫也不会死磕埋点。URL 本身不可能被加密，所以加密的就只有报文了。当然，弱加密只是相对而言，如果你的对手真的死磕你的埋点，你也可以提升加密等级，这个不能认死理。

那我们就来看看，埋点信息埋什么呢？这里主要分这么几类：

第一类，常规用户访问数据；

第二类，爬虫访问数据，包括爬虫类型，来源，数量，等等；

第三类，部分测试策略的线上效果收集。这个在 13 讲会详细说明；

有了这些数据之后，我们以后做效果检测，或者说，制作 ROI 报表就容易了很多。严格来说，埋点统计算是 BI 的一部分，虽然只是出了几个报表。

实际上，这类报表的汇报意义远远大于实际意义。我在前面说过，我们的系统做了实时的熔断、实时的监控，因此埋点通常只是一些统计数据，用来衡量业务指标的。

但是要注意，汇报也是工作的一部分。在进阶篇的时候我们会提到相关技巧。

例如，所有项目都要汇报 ROI，反爬也不例外。没有 ROI 支撑的项目，光靠理想是活不下去的。而 ROI 的来源，就是这些埋点数据。

埋点统计的痛点

排除掉加密信息这个点之后，**埋点信息有几个最大的痛点：并发压力，数据存储压力以及读取速度。**

因为通常来说，这三样同时实现，几乎是不现实的。因此如果必须舍弃一样的话，我们会**选择舍弃读取速度。**因为如果能砍掉产品经理提出的“实时查看爬虫状态”这种无意义的需求的话，我们就可以完成这一切了。为什么说这个需求是无意义的呢？因为实时爬虫状态对于做出决策没有任何价值。有爬虫，系统拦截即可；无爬虫，系统运行即可。看得到，看不到，对公司没有任何影响。

这个主要的影响反倒是在技术侧，技术如果发现误伤较高，系统又没有自动熔断，那么需要手动熔断。当然，这个在误伤统计收集里面就做掉了。而误伤统计本身又没有高并发问题，相当于砍掉了并发压力，来实现需求。

最后，怎么砍掉这个需求呢？砍，也是有技巧的，你凭空砍掉产品经理一个需求，严重影响他的 KPI，这是他难以接受的，也必然会与你发生争议。而你给他换个需求，就像上面一样，把埋点监控换成误伤统计监控，他能给老板交差了，也就自然不会和你死磕。

埋点统计的实现方式

因此我们的实现方式就很好理解了：前端使用大量 Server 换取网络链接数，同时聚合数据，避免冲击后端收集模块。后端使用常规的高写入速度、低读取速度的存储方式来承接即可。大公司通常有成熟的系统，但是小公司往往没有这个模块，需要自己来实现。

读取数据的时候，因为是高写入速度、低读取速度，我们的读取速度用来换高并发和写入速度了，那就意味着不可能实时查看。而上面我们已经和产品砍掉了实时查看的需求，那么我们的查看方式就只有每日报表这一种了。**实际上，每日一次的报表足够使用了，如果真的有比较实时的要求，撑死也就是一个小时，再高就没有实际意义了。**

低读取速度的存储，应对一小时一次的读取，也是没有任何压力的。这个完全不用担心。

规则引擎

综合上一讲说到的 key 的加密以及这一讲的信息收集、埋点统计分析，你可能意识到了一个问题：我们线上需要大量的变更，而变更就要发布，发布就有流程。但是爬虫不等人，等你变更发布完可能一天过去了，爬虫都走人了。所以，**我们需要一个能够实时变更生产规则的系统。这个系统，我们就称为它规则引擎。**

那么在反爬虫这个 case 里，**规则引擎有三个主要构成部分，分别是信息收集模块、规则判断模块以及规则执行模块。**其中，规则判断一般是在服务端或者 BFF 实现的，而信息收集和规则执行都会在前端执行。因此我们主要来看这两点。

首先看信息收集模块的设计。

严格来说，**信息收集模块是流程的起点，因此对于状态机而言它没有前置状态。**

如前面所说，客户端收集的信息，会在客户端进行加密，然后传输过来，那么就可能是可逆加密与不可逆加密两种情况。因此，这里可能会有解密，也就是说会使用到对称加密架构，但也可能没有解密却有验证模块，这就会使用到不可逆加密了，不过这样做的缺点就是可能会丢失细节。

此外，**规则执行模块，这里依然取决于你面临的状况。**

有的系统，针对反爬虫，是无脑拦截的，那么前端无需做什么事情，针对服务的拦截响应做一个友好的兼容处理即可。但是有的系统设计，比较重前端，会进行很多操作，比如前面提到的 Cookie 等等。这个时候，规则引擎不再是状态机，这里是纯粹的 rule。客户端只要写 rule 的解释逻辑即可。

这里需要注意的是，服务端如果增加了 rule，可能导致客户端不兼容，需要考虑热更新的情况。例如，服务端增加了 rule，CDN 也发布了，但是客户已经几天没刷过页面了，用的还是三天前的 js 代码。这时候需要考虑做客户端版本容差，单独拉取 rule 规则执行。

最后，针对规则引擎部分，我们可以做个总结。其实面对“不等人”的爬虫，规则引擎存在的主要作用有四点：

第一，降低发布时间，提升响应速度。

第二，降低研发成本，避免每次都写代码变更。

第三，提升灵活性，更敏捷的对爬虫做出反应。

最后，基于规则而不是基于代码片段特性，可以降低出错的概率。

那么到这里，前端，这个反爬虫的主战场，能够做的事情就已经介绍完了。

小结

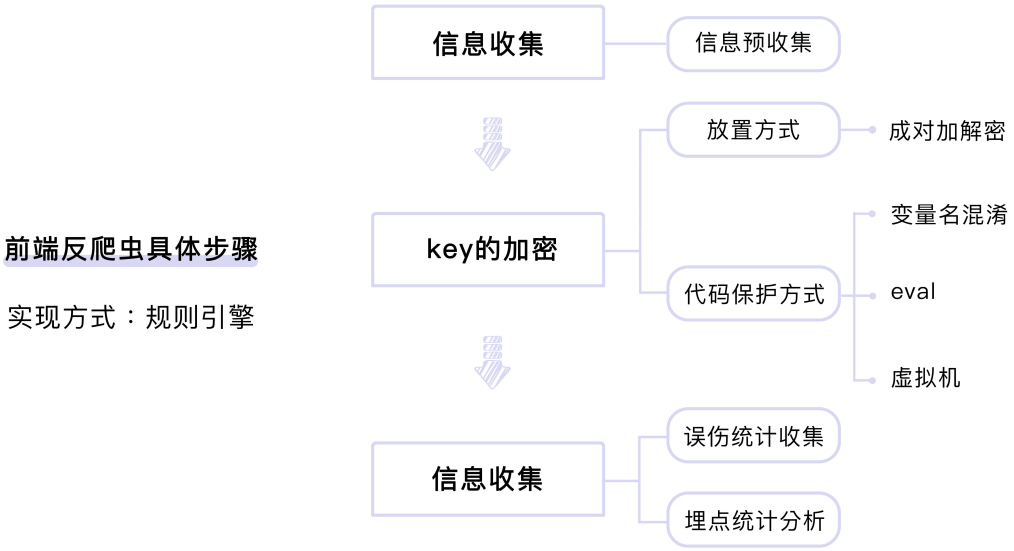
最后，我来给你做个总结吧。这一讲，我们讲解了前端的辅助工作，信息收集以及规则引擎。

信息预收集本质上相当于游戏里面的插眼，帮助你有更好的视野来进行决策。但是我们都知道，不能把眼镜插在太明显的地方，否则很容易被人排掉，这就是加密的意义所在。而完成预收集之后，就是我们在上一讲一直讨论的 key 加解密以及混淆的过程了。

接着，在反爬虫动作完成后，不论是出于对效果的检测还是对自己的保护，我们都会进行误伤统计以及埋点统计的分析.。由于这个数据会用于复盘，对自己的决策十分重要，因此如果被爬虫知道了，他们一定会爬这个埋点统计接口来干扰你。所以，这个加密程度也可以根据情况适当调高一些。

最后，规则引擎帮助我们快速响应变化，避免贻误战机。

以上，前端反爬虫的所有动作，就算完成了。在文字的下方，我也为你准备了一张前端反爬虫的完成动作图，希望帮助你理解这两讲的关系。



思考题

好了，又到了愉快的思考题时间。还是三选一的老规矩，你可以任选一个问题在留言区和我一起讨论。

1. 规则引擎本质上跳了生产发布，而制定生产发布流程是为了降低线上变更风险。那么，如果测试质疑你在引入风险越过测试权限，如何回应呢？
2. 理论上说信息收集是越多越好的，但是过多了又会对系统造成较大的压力。那么如何评估需要收集哪些数据呢？
3. 收集的数据里不能包含用户隐私数据， 否则可能有法务风险，尤其是跨国项目。 那么，如何保证自己收集的数据不会不小心包含用户隐私呢？

期待你在评论区的分享，我会及时回复你。今天诗句的下方，也有关于规则引擎的实现方式——状态机的探讨，如果你有精力，也可以和我一起在评论区讨论。反爬无定式，我们一起探索。

极客时间

出兵之前先插眼，打团过后要复盘。

反爬虫兵法演绎 20 讲

DS Hunter / 反爬虫专家，前某公司研发总监

识别二维码
免费试读

规则引擎的实现方式：状态机

有些课程可能会用状态机这个词，其实规则引擎和状态机基本是一个意思。确切来说，大部分规则引擎是用状态机实现的。没做过的人可能觉得状态机这个词有点高大上，但是说穿了，状态机也不是什么神秘的东西。

现在正好是年初，也许你正在刷题准备面试，那么你一定会做到 DP 相关的题目，所有题解都会和你说：DP 的核心是找到状态转移方程。注意这个词：状态转移。这个词虽然是讲解 DP 的，但却非常清晰地描述了**状态机的核心：状态，以及转移。**状态，state，名词；转移，也就是 action。你的系统能支持这两个事情基本足够了。

简单地说，你的系统存在各种节点（state），在每个 state 下，根据不同的条件，可以进入另一个状态。也就是 action。

有了状态机，就可以实现任意的流程图。这是规则引擎的重要作用。实际上，我们写的很多代码，本质上只是在写流程，例如流量进来，进行规则判定，根据判定结果，进行不同的处理。看起来是代码逻辑，但是抽象一下，这就是流程。看起来是流程，但是再抽象一下，就是状态机。因此有了状态机，就可以任意实现流程变更，而不需要发布了。因此，能大大节约时间，提升灵活性，降低出错概率。

状态机可能在服务端执行，也可能在 BFF 执行，这取决于公司的架构。当然，要尽量适应公司架构，而不是去改变架构，因为反爬并非一个高 ROI 项目，不具备这样的实力。如果公司没有 BFF，那就老老实实在服务端做就可以了。

分享给需要的人，Ta订阅超级会员，你将得 50 元

Ta单独购买本课程，你将得 20 元

生成海报并分享

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | 反爬虫概述（二）：无收益的前端是怎么进行key处理的呢？

下一篇 答疑课堂 | 爬虫反爬虫调试对抗技巧以及虚拟机详解

精选留言 (1)

写留言

peter
2022-02-22

请教老师几个问题：
Q1：爬虫一般用什么语言开发？
是不是主要是用Python？用Java可以吗？
Q2："前端使用大量 Server 换取网络链接数"，server怎么理解？

前端不就是网页吗？难道在网页里加了“server”？...

展开

作者回复: 第一个，语言无所谓，都一样。第二个，前端会有bff，在serverless还没完全一统天下之前，前端还是在处理服务器。这也就是nidejs当年出来的愿景之一。第三个，硬盘速度不变，但是不同数据结构会对读写有影响，一般优化读速度都要牺牲写速度，反之亦然。不过一般不至于自己写数据一。举个例子，无索引的表读取很慢，但是写入速度爆快，也不是无优点的。还有更多别的选择，大数据有专家研究这个。第四个，js会被破解，但是破解要时间，破解速度低于改版速度就等于不会被破解。举个例子，你每天上班，担心家里被盗。所有的锁都能破解，但是如果破解需要48小时，那等于永远无法破解，因为你24小时肯定会回。